IE 529 HW3 Zhenye Na Zna2

1. Use induction to prove *Jensen's Inequality*, as follows:
$\sum_{i=1}^{n}\alpha_i f(x_i) \le f\left[\sum_{i=1}^{n}\alpha_i x_i\right]$

Base case:

For $n=1,2$, the equality is true.

$$\alpha_1 f(x_1)+\alpha_2 f(x_2) \le f(\alpha_1 x_1+\alpha_2 x_2)$$

Induction steps:

Assume that $n=m$, the inequality stays true.

$$\sum_{i=1}^{m}\alpha_i f(x_i) \le f\left(\sum_{i=1}^{m}\alpha_i x_i\right)$$

Then for $n=m+1$:

$$f\left(\sum_{i=1}^{m}\alpha_i x_i\right) = f\left[\alpha_{m+1}x_{m+1} + \sum_{i=1}^{m}\alpha_i x_i\right)$$
$$= f\left[\alpha_{m+1}x_{m+1} + (1-\alpha_{m+1})\sum_{i=1}^{m}\frac{\alpha_i}{1-\alpha_{m+1}}x_i\right)$$
$$\ge \alpha_{m+1}f(x_{m+1})+(1-\alpha_{m+1})f\left[\sum_{i=1}^{m}\frac{\alpha_i}{1-\alpha_{m+1}}x_i\right.$$

as α_{m+1} is not related to i, so $\rightarrow$ $\alpha_{m+1}f(x_{m+1}) + f\left[\sum_{i=1}^{m}\alpha_i x_i\right]$

$$\ge \alpha_{m+1}f(x_{m+1}) + \sum_{i=1}^{m}\alpha_i f(x_i)$$
$$= \sum_{i=1}^{m+1}\alpha_i f(x_i)$$

Equality holds if and only if $x_1=x_2=\cdots\cdots=x_n$, or $f(x)$ is linear.

2. We have noticed that $f(x)=\log(x)$ is concave on $(0,\infty)$
So, we take log on the left-hand side of the inequality.
$\log\left(\prod_{i=1}^{n}x_i\right)^{\frac{1}{n}} = \frac{1}{n}(\log x_1+\log x_2+\cdots\cdots+\log x_n) = \sum_{i=1}^{n}\frac{1}{n}\log x_i \le \log\left(\sum_{i=1}^{n}\frac{1}{n}x_i\right)$
Because $\{x_i\}$ is a non-negative set, $f(x)=\log(x)\nearrow$
So, $\left(\prod_{i=1}^{n}x_i\right)^{1/n} \le \left(\frac{1}{n}\sum_{i=1}^{n}x_i\right)$ ⁂

3. $Y=\beta x+e$

(a)

$$Y=\begin{bmatrix}y_1\\y_2\\y_3\\\vdots\\y_n\end{bmatrix} \quad X=\begin{bmatrix}x_1\\x_2\\x_3\\\vdots\\x_n\end{bmatrix} \quad \text{and} \quad e=\begin{bmatrix}e_1\\e_2\\e_3\\\vdots\\e_n\end{bmatrix}$$

and $\beta = [\beta]$, $Y = X\beta + e$

The least-squares line minimizes

$$Q(\beta) = \sum_{i=1}^{n} (y_i - \beta x_i)^2 = (Y - X\beta)^T (Y - X\beta)$$

The least squares estimate $\hat{\beta}$ solves the first order equation : $\frac{\partial Q(\beta)}{\partial \beta} = 0$

and is given by

$$\hat{\beta} = (X^T X)^{-1} X^T Y = \sum_{i=1}^{n} (x_i^2)^{-1} \sum_{i=1}^{n} x_i y_i = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2}$$

(b) $\hat{\beta} = \sum_{i=1}^{n} w_i y_i$, where $w_i = \frac{x_i}{\sum_{j=1}^{n} x_j^2}$. It can be considered as a weighted sum of the independent normal random variables : $y_i \sim N(x_i\beta, \sigma^2)$, which is a normal distribution. So next step is to figure out μ_β and σ_β^2.

$$\begin{aligned} E(\hat{\beta}) &= E\left(\sum_{i=1}^{n} w_i y_i\right) \\ &= \sum_{i=1}^{n} w_i E(y_i) = \sum_{i=1}^{n} w_i \cdot (x_i \beta) \\ &= \sum_{i=1}^{n} \frac{x_i}{\sum_j x_j^2} \cdot x_i \beta \\ &= \beta \cdot \sum_{i=1}^{n} \frac{x_i^2}{\sum_j^n x_j^2} = \beta \end{aligned}$$

$$\begin{aligned} Var(\hat{\beta}) &= Var\left(\sum_{i=1}^{n} w_i y_i\right) \\ &= \sum_{i=1}^{n} w_i^2 \cdot Var(y_i) = \sum_{i=1}^{n} w_i^2 \cdot \sigma^2 \\ &= \sigma^2 \times \sum_{i=1}^{n} \left[\left(\frac{x_i^2}{\sum_{j=1}^{n} x_j^2}\right)^2\right] \\ &= \sigma^2 \times \frac{\sum_{i=1}^{n} x_i^2}{(\sum_{j=1}^{n} x_j^2)^2} \\ &= \sigma^2 \times \frac{1}{\sum_{j=1}^{n} x_j^2} \end{aligned}$$

So, $\hat{\beta} \sim N(\beta, \sigma_\beta^2)$ where $\sigma_\beta^2 = \sigma^2 \times \frac{1}{\sum_{j=1}^{n} x_j^2}$

(c). $SS_R = \sum_{i=1}^{n} (y_i - x_i\hat{\beta})^2 \sim \sigma^2 \chi^2_{(n-1)}$

(d)

A significance level γ test of H_0 :

reject H_0 if $\sqrt{\frac{(n-2) S_{xx}}{SS_R}} |B| > t_{\frac{\gamma}{2}, n-1}$

accept H_0 otherwise

rejecting H_0 if the desired significance level is at least as large as:

$$\begin{aligned} p\text{-value} &= P\{|T_{n-1}| > v\} \\ &= 2P\{T_{n-1} > v\} \end{aligned}$$

Problem 4:

Ex4

```
SurvivalTime, MismatchScore, Age
624, 1.32, 51
46, 0.61, 42.5
64, 1.89, 54.6
1350, 0.87, 54.1
280, 1.12, 49.5
10, 2.76, 55.3
1024, 1.13, 43.4
39, 1.38, 42.8
730, 0.96, 58.4
136, 1.62, 52.0
836, 1.58,45.0
60, 0.60,64.5
```

Below is the R program used for this problem.

```r
# read csv file
mydata <- read.csv("/Users/macbookpro/Desktop/Ex4.csv")
x1<- mydata$MismatchScore;
x2<- mydata$Age;
t <- mydata$SurvivalTime;
y <- log2(t);

#fit log model
fit <- lm(y ~ x1 + x2)

#Results of the model
summary(fit)
```

```
Call:
lm(formula = y ~ x1 + x2)

Residuals:
   Min     1Q Median     3Q    Max 
-3.481 -1.815  0.166  1.799  2.402 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)  
(Intercept)  11.5724     5.3701    2.15     0.06 .
x1           -1.6578     1.1399   -1.45     0.18  
x2           -0.0366     0.1004   -0.36     0.72  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.28 on 9 degrees of freedom
Multiple R-squared:  0.201,	Adjusted R-squared:  0.0231 
F-statistic: 1.13 on 2 and 9 DF,  p-value: 0.365
```

(a) Let the dependent variable be the logarithm of Survival time. Fit a multiple linear regression on the independent variables of Mismatch score and Age.

Solution: Based on the program, we can see the β_0 **equals 11.5724**. The estimate of **Mismatch score (x1) equals -1.6578**. The estimate of **Age (x2) equals -0.0366**.

(b) Compute an estimate of the variance of the error term.

Solution: The residual standard error term will be 2.277 (it is different from the result in screenshot because after several times of running program, the answer is 2.277, not 2.28).

Problem 5:

Ex5

```
AngerScore, SecondHeartAttack
80, Yes
77, Yes
70, No
68, Yes
64, No
60, Yes
50, Yes
46, No
40, Yes
35, No
30, No
25, Yes
```

Below is the R program used for this problem.

```r
# read csv file
mydata <- read.csv("/Users/macbookpro/Desktop/Ex5.csv")
y <- mydata$SecondHeartAttack;
x <- mydata$AngerScore;

# change "YES" or "NO" into booleans
for (i in mydata$SecondHeartAttack){
   if (i == "Yes"){
      i <- 1
   }
   else{
      i <- 0
   }
}

# perform logistic regression on dataset
mylogit <- glm(y ~ x, data = mydata, family = "binomial")
# show the summary
summary(mylogit)

# predict the new value
newdata <- data.frame(x = 55 )
pred <- predict.glm(mylogit, newdata, type = "response")
pred
```

```
Call:
glm(formula = y ~ x, family = "binomial", data = mydata)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5201  -1.1511   0.7920   0.9932   1.3494

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.04733    1.88556  -0.555    0.579
x            0.02606    0.03406   0.765    0.444

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 16.301  on 11  degrees of freedom
Residual deviance: 15.690  on 10  degrees of freedom
AIC: 19.69

Number of Fisher Scoring iterations: 4

> |
```

(a) Explain how the relationship between a second heart attack and one's anger score can be analyzed via a logistic regression model.
Solution: Based on the dataset, the values of "Second Heart Attack" contain 'Yes' and 'No'. They can be categorized as two categories. So to some extent, it is a 'Binary Classification' problem. So it can be solved via logistic regression.

(b) Using a software package of your choice, estimate parameters for this model (for example, in Matlab to fit a logistic model consider the command 'glmfit').

(c) Estimate the probability that a heart attack patient with an anger score of 55 will have a second heart attack within 5 years.
Solution: Based on the program, the probability is 0.5953763.

```
> newdata <- data.frame(x = 55 )
> pred <- predict.glm(mylogit, newdata, type = "response")
> pred
        1
0.5953763
```

Problem 6:

(a) For this data set compute the SVD (singular value decomposition) of the original matrix, and using this SVD discuss the expected results of performing a PCA on this data.
Basically the SVD on scaled matrix is the same as PCA. So the SVD on original matrix, I think it is helpless for the PCA.

(b) Compute the PCA: First compute the mean(s) for the data, and subtract from the original data; second compute the covariance matrix including the scaling 1/(n – 1); third compute an eigenvalue decomposition and sort both the eigenvalues and eigenvectors in descending order.

(c) Plot and discuss the principal components. Discuss how this process and results might differ from a direct SVD of the de-biased, scaled data.

Solution: The covariance matrix C is given by $\mathbf{C} = \mathbf{X}^T\mathbf{X}/(\mathbf{n}\text{-}\mathbf{1})$. It is a symmetric matrix so it can be diagonalized as: $\mathbf{C}=\mathbf{V}\mathbf{L}\mathbf{V}^T$, where **V** is a matrix of eigenvectors (each column is an eigenvector) and **L** is a diagonal matrix with eigenvalues ***lamda*** in the decreasing order on the diagonal.

With Singular Decomposition of X, we can get $\mathbf{X}=\mathbf{U}\mathbf{S}\mathbf{V}^T$, where S is the diagonal matrix of singular values s_i. So we can use some matrix multiplication rules to get

$$\mathbf{C} = \mathbf{V}\mathbf{S}\mathbf{U}^\top\mathbf{U}\mathbf{S}\mathbf{V}^\top/(n-1) = \mathbf{V}\frac{\mathbf{S}^2}{n-1}\mathbf{V}^\top,$$

With the eigenvalue decomposition, $\mathbf{C}=\mathbf{V}\mathbf{L}\mathbf{V}^T$, so we can obtain ***lamda*** = $s^2/(\mathbf{n}\text{-}\mathbf{1})$. So we can know why this process might differ from SVD.

In [6]:
```
# read data and import packages

import math
import numpy as np
from scipy import linalg
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

raw_matrix = numpy.loadtxt(open("/Users/macbookpro/Desktop/IE 529 HW3/Dat
```

In [5]:
```
#perform SVD on original matrix

U,S,V = linalg.svd(raw_matrix)
```

In [13]:
```
#perform matrix scaling and center the data in oringinal matrix
n = 200
centered_matrix = (raw_matrix - raw_matrix.mean(axis=1)[:, None])/ math.s
cov = np.dot(centered_matrix, centered_matrix.T)

#calculate the eigenvalue and eigenvectors
e, v = np.linalg.eig(cov)

#sort the eigenvalues
sorted_e = sorted(e , reverse = True)

vT = v[:,[0,2]]

print (sorted_e)
print (e)
print (v)
print(vT)
```

```
[[ 0.66943821  0.22771139]
 [ 0.32203254 -0.9467286 ]
 [ 0.66943821  0.22771139]]
[2.3829744914056272, 0.23466776481580504, 7.1043968626113282e-17]
[  2.38297449e+00   7.10439686e-17   2.34667765e-01]
[[  6.69438214e-01   7.07106781e-01   2.27711392e-01]
 [  3.22032538e-01   5.40520479e-16  -9.46728601e-01]
 [  6.69438214e-01  -7.07106781e-01   2.27711392e-01]]
```

In [11]:
```
renew_data= np.dot(vT.T, centered_matrix)
print(renew_data)
```

```
[[ -1.48883676e-01  -1.01591685e-01  -4.89669133e-02  -1.53447482e-01
   -1.92176620e-01  -1.51254286e-01  -8.55679611e-02  -7.24621513e-02
   -8.33180676e-02  -1.15239006e-01  -7.47009822e-02  -1.16565833e-01
   -5.36900250e-02  -1.48188761e-01  -6.55905856e-02  -1.30247568e-01
   -9.55914016e-02  -5.71234454e-02  -4.19264761e-02  -1.54887272e-01
   -4.15845085e-02  -4.90316846e-02  -1.15587637e-01  -1.03512955e-01
   -1.26540238e-01  -9.67062443e-02  -8.30413808e-02  -9.15261532e-02
   -5.55801259e-02  -4.98841910e-02  -6.51740725e-02  -1.28946615e-01
   -9.07731440e-02  -8.43626303e-02  -1.53663125e-01  -4.42292880e-02
   -7.16734325e-02  -7.77625621e-02  -7.74096271e-02  -7.64352488e-02
   -1.37499432e-01  -9.22453570e-02  -9.47806040e-02  -1.08969066e-01
   -1.56463733e-02  -1.37207201e-01  -1.05426169e-01  -1.51997380e-01
   -1.55451769e-01  -1.17595155e-01  -1.11342905e-01  -1.39598640e-02
   -9.63610580e-02  -1.56057016e-01  -2.32566078e-02  -3.38176094e-02
   -1.08197555e-01  -1.61999919e-01  -1.10503439e-01  -1.11984418e-01
   -8.28903941e-02  -1.30698983e-01  -5.98541574e-02  -8.25572392e-02
   -1.67061647e-01  -1.41853711e-01  -1.45445316e-01  -1.18306364e-01
   -1.15582203e-01  -6.85233970e-02  -2.24726093e-01  -1.44157280e-01
   -2.99729943e-02  -1.60548604e-01  -5.25671666e-02  -7.54647646e-02
```

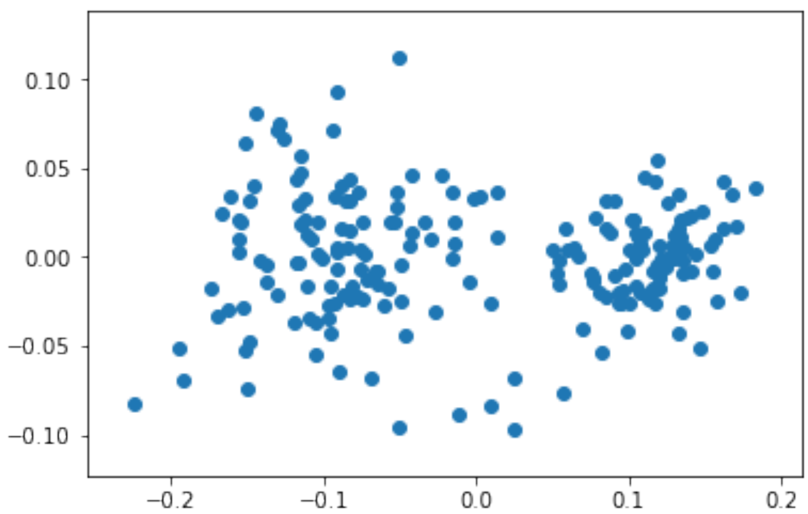
```
  -8.68102642e-02  -8.84302427e-02  -1.50246567e-01  -1.03470938e-01
  -1.41121171e-02  -8.17118432e-02  -9.10324265e-02  -1.18753421e-01
  -9.17430768e-02  -9.18912492e-02  -9.43414457e-02  -9.04465402e-02
  -1.12091122e-01   3.24104153e-03  -1.94712077e-01   1.31237494e-02
  -8.82752775e-02  -5.18529624e-02  -1.74368029e-01  -1.05088519e-01
  -1.16990475e-01  -6.49499568e-02  -1.69706857e-01  -7.42589880e-02
   9.97876132e-02   1.07682616e-01   1.20247960e-01   7.64139004e-02
   1.10308288e-01   8.85399755e-02   1.20189464e-01   1.27879387e-01
   8.52597283e-02   1.06180405e-01   1.30355025e-01   5.96023794e-02
   1.54855026e-01   1.83320354e-01   1.58383722e-01   1.02295210e-01
   1.32250204e-01   7.87648837e-02   1.67196110e-01   1.41399881e-01
   1.43845961e-01   1.00188402e-01   1.29847821e-01   1.00227609e-01
   1.15398754e-01   1.07921846e-01   9.02997517e-02   9.67197744e-02
   1.15526375e-01   6.35442891e-02   1.41184329e-01   1.10827516e-01
   1.34445825e-01   1.27570027e-01   1.30647758e-01   7.55522222e-02
   9.37046385e-02   1.13120182e-01   1.20446145e-01   1.53060284e-01
   1.08790161e-01   1.56176059e-01   1.35227152e-01   1.47881531e-01
   1.24179012e-01   1.04098175e-01   7.71337379e-02   1.34950295e-01
   1.37821149e-01   1.05299122e-01   9.13002711e-02   1.16125289e-01
   9.33916945e-02   1.04094459e-01   1.19687668e-01   1.21247193e-01
   1.18857428e-01   1.36860043e-01   1.24805987e-01   1.71026112e-01
   1.46652217e-01   1.62064710e-01   8.53941452e-02   1.25863602e-01
   1.03779410e-01   1.32793510e-01   1.34439802e-01   5.49397936e-02
   8.46030605e-02   8.03161540e-02   1.35271284e-01   1.72723472e-01
   1.17754065e-01   1.33020376e-01   1.18266309e-01   9.49126823e-02
   1.17421380e-01   9.77243630e-02   1.62096327e-01   1.33003327e-01
  -1.04004123e-03  -4.17906333e-03   5.28577435e-02   5.90738465e-02
   1.37434103e-02   9.54715583e-03  -4.58686709e-02   5.78160665e-02
  -2.63471696e-02  -5.04175573e-02   6.74522304e-02  -1.09677198e-02
   2.44824649e-02   5.38200219e-02  -1.59145932e-02   8.18919854e-02
   2.42945140e-02   5.07088304e-02   7.03207919e-02   1.00154635e-02]
 [ 3.12905781e-02  -5.08198340e-04  -4.55415119e-03  -2.79110515e-02
  -6.93733823e-02  -5.25857434e-02   3.17894380e-02   2.25704085e-03
   4.35421851e-02   4.68134579e-02  -2.35059073e-02  -3.12327319e-03
   2.03202394e-02  -4.79938299e-02  -8.04479961e-03  -2.08881041e-02
  -4.27943200e-02  -1.77140484e-02   1.32877827e-02   9.68759074e-03
   4.67752491e-02  -2.47854637e-02   5.75542696e-02   2.23244689e-03
   6.64457702e-02  -3.45701791e-02  -2.39842956e-02   5.34912202e-03
   2.02733363e-02   1.12641178e-01  -7.40144608e-03   7.47467183e-02
   9.24804565e-02   4.83566496e-03   1.97676029e-02   6.08003479e-03
  -1.31988270e-02  -2.20389360e-02   3.60717919e-02  -6.29757962e-03
  -1.40102079e-02  -2.57747585e-02  -1.57874648e-02  -3.41514592e-02
   3.61444104e-02  -4.63355442e-03  -3.70645628e-02   6.43309901e-02
   2.54025840e-03   4.40959063e-02   1.29765271e-02   7.49586650e-03
  -2.71672387e-02   2.14414740e-02   4.62518969e-02   1.98947114e-02
   1.06500101e-02  -2.98767390e-02  -1.60365157e-02   3.32513784e-02
   1.49521444e-02   7.18916088e-02  -2.67146400e-02   3.19973856e-02
   2.47703411e-02  -2.21789749e-03   4.05393657e-02  -2.93435375e-03
   1.82015583e-02  -6.83149729e-02  -8.25342481e-02   8.09707548e-02
   9.80620157e-03   3.42600647e-02   2.81951163e-02   4.70249437e-03
  -2.09983081e-02   1.66693282e-02  -7.35978620e-02   1.93366468e-02
   1.94618226e-02  -1.61566730e-02   2.37223403e-03  -3.68506704e-02
  -6.77984909e-03   3.46268532e-02   7.20268392e-02  -6.38218388e-02
   2.10318592e-02   3.37582535e-02  -5.07902896e-02   3.67380136e-02
   4.03711661e-02   3.63997178e-02  -1.81095464e-02  -5.45907122e-02
   2.89600581e-02  -1.49401951e-02  -3.27528582e-02   2.00228837e-02
  -4.11565178e-02   7.82538970e-03  -1.06855178e-02  -1.09498777e-02
   4.45920513e-02   1.38258968e-02  -1.12295538e-02   6.38404318e-03
   1.57800647e-02   1.18369738e-02   3.82388308e-04   4.22978160e-03
  -7.57708120e-03   3.83983268e-02  -2.51109898e-02   2.07061620e-02
   2.15768059e-03   2.26084681e-02   3.47780526e-02   2.30757017e-02
   1.56447024e-03  -2.54308140e-02  -9.28267808e-04   4.12146689e-03
  -1.74116446e-02  -1.95880401e-02  -1.03373071e-02  -1.82843645e-02
  -8.32310220e-03   5.43585542e-03  -7.39016057e-03   1.41140578e-02
   2.09498648e-02   1.06737982e-03   1.30013832e-02  -9.69302791e-03
  -1.98490857e-02  -2.33831657e-02   7.06782345e-03   6.35352954e-03
```

```
   4.00901696e-03   9.61475099e-03  -2.23581526e-03   2.58120801e-02
  -2.28746541e-03  -1.64409843e-02  -1.41867996e-02  -3.13073564e-02
   2.22028192e-02   1.40231167e-02   3.20164616e-02  -1.92985039e-02
  -2.60580034e-02  -1.14489364e-03  -1.78171933e-02  -1.23310457e-03
  -4.44953945e-03   5.66504327e-03  -5.78060561e-03   1.75413141e-02
  -5.10778719e-02   4.29837345e-02   3.23511229e-02   3.00857802e-02
   2.15105247e-02   1.65616145e-02   1.11614030e-02  -1.47501190e-02
  -2.17320083e-02  -1.97758248e-02  -9.21430786e-03  -1.95444140e-02
  -2.57679258e-02   3.57249101e-02   5.41951600e-02  -2.63268425e-02
   4.28259711e-02  -6.60628519e-03   1.59938722e-02  -4.31412599e-02
   3.32379920e-02  -1.44406376e-02  -9.03389486e-03   1.61501097e-02
   1.16928084e-02  -8.32782945e-02  -4.34545987e-02  -7.67507225e-02
  -3.10188287e-02  -9.57984175e-02   4.86936316e-05  -8.82797396e-02
  -6.82243994e-02  -1.68311804e-03  -1.23328459e-03  -5.39826913e-02
  -9.73705266e-02   4.72382976e-03  -4.00754973e-02  -2.57557909e-02]
]
```

In [15]:

```
# plot 1
X1 = renew_data[0]
X2 = renew_data[1]

plt.scatter(X1, X2)
plt.show()
```



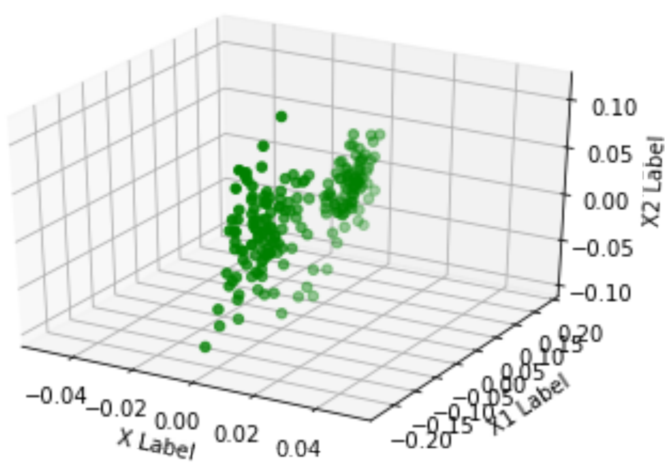
In [21]:
```
#plot 2
plot = plt.figure()
ax = plot.add_subplot(111, projection='3d')

x = np.zeros(200)
y = X1
z = X2

ax.scatter(x, y, z, c='g', marker='o', label = 'PCA dibased', )

ax.set_xlabel('X Label')
ax.set_ylabel(' X1 Label')
ax.set_zlabel('X2 Label')

plt.show()
```



In [27]:
```
# perform SVD on scaled matrix

reconstruct_matrix = centered_matrix
U1, S1, V1 = linalg.svd(reconstruct_matrix)
Ans = np.dot(U1.T, reconstruct_matrix)
print (Ans)
```

```
[[  1.48883676e-01   1.01591685e-01   4.89669133e-02   1.53447482e-01
    1.92176620e-01   1.51254286e-01   8.55679611e-02   7.24621513e-02
    8.33180676e-02   1.15239006e-01   7.47009822e-02   1.16565833e-01
    5.36900250e-02   1.48188761e-01   6.55905856e-02   1.30247568e-01
    9.55914016e-02   5.71234454e-02   4.19264761e-02   1.54887272e-01
    4.15845085e-02   4.90316846e-02   1.15587637e-01   1.03512955e-01
    1.26540238e-01   9.67062443e-02   8.30413808e-02   9.15261532e-02
    5.55801259e-02   4.98841910e-02   6.51740725e-02   1.28946615e-01
    9.07731440e-02   8.43626303e-02   1.53663125e-01   4.42292880e-02
    7.16734325e-02   7.77625621e-02   7.74096271e-02   7.64352488e-02
    1.37499432e-01   9.22453570e-02   9.47806040e-02   1.08969066e-01
    1.56463733e-02   1.37207201e-01   1.05426169e-01   1.51997380e-01
    1.55451769e-01   1.17595155e-01   1.11342905e-01   1.39598640e-02
    9.63610580e-02   1.56057016e-01   2.32566078e-02   3.38176094e-02
    1.08197555e-01   1.61999919e-01   1.10503439e-01   1.11984418e-01
    8.28903941e-02   1.30698983e-01   5.98541574e-02   8.25572392e-02
    1.67061647e-01   1.41853711e-01   1.45445316e-01   1.18306364e-01
    1.15582203e-01   6.85233970e-02   2.24726093e-01   1.44157280e-01
    2.99729943e-02   1.60548604e-01   5.25671666e-02   7.54647646e-02
```

In [ ]: